

# Mise en place d'une solution décisionnelle

GUILLAUMOT Nadia

MOUMNI Rim

MANYRI Colin

TRELUYER Daniel

RHAFIRI Lojaïn

MOUAYED Dina

PHAM Estelle



**Quoi ?** Développer un Système d'Information Décisionnel (SID).

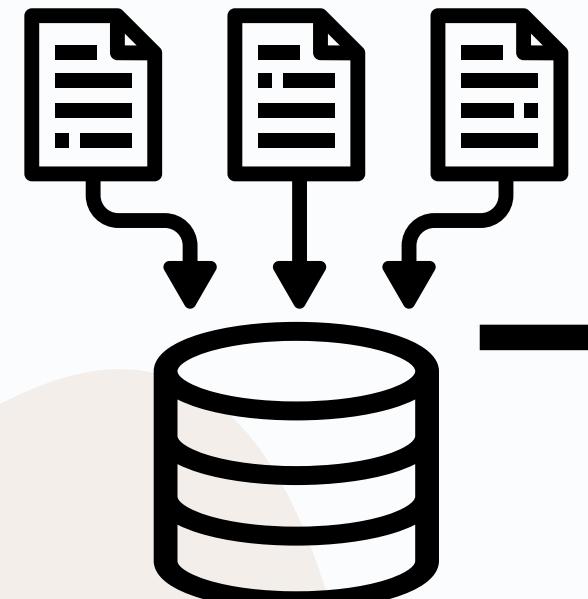
- **Pourquoi ?** Suivre l'activité d'un établissement de santé.
- **Comment ?** A travers une chaîne de traitement de donnée, de l'ingestion des données brutes jusqu'à la visualisation des KPI sur PowerBi.
- **Quels contraintes ?** Outils imposés : Apache Airflow et Snowflake, PowerBi
- **Deadline ?** Un mois.

**Outils imposés**



# Introduction - Pipeline Générale

**Data Hospital**  
fichiers textes journaliers



**Staging (STG)**



Lot 1

Installation de  
l'environnement de travail  
Conception de la solution

**Technique (TCH)**



Ajout dans WRK →

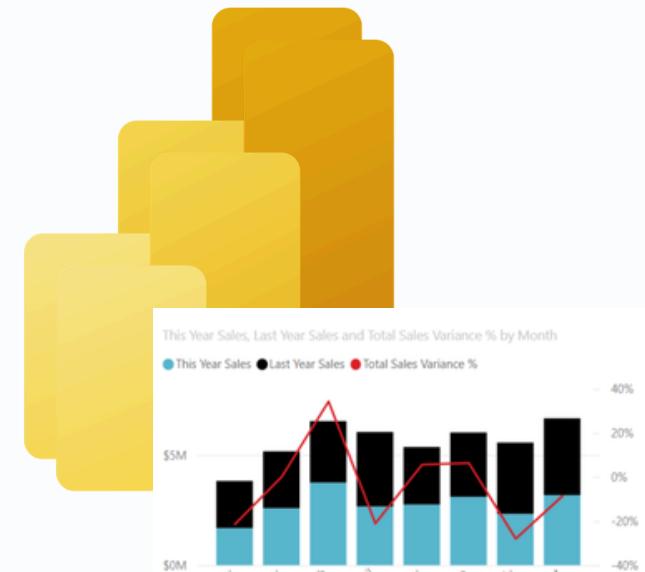


**Work (WRK)**

Ajout dans SOC



**Socle (SOC)  
Vues**



**Visualisation via  
PowerBI (PBI)**

Répétitions journalière

**Créations des tables  
Alimentation de STG**

Lot 2

Conception de la solution

**Tableau de bord PBI  
Réponses aux questions posées**

Lot 3

Passage de STG à socle  
Mise en place de DAGs Airflow pour les répétitions journalières.

Lot 4

# 1. Mise en place de l'environnement

**Snowflake** : Gestion des données

**VSCode** : IDE (Python, SQL, md)

**GitLab** : Versionning + Code Collaboratif

**Taiga** : Gestion de projet

**Teams** : Communication entreprise

**Messenger** : Communication interne



Livrable 2 : Script sql 27 May 2025 to 03 Jun 2025

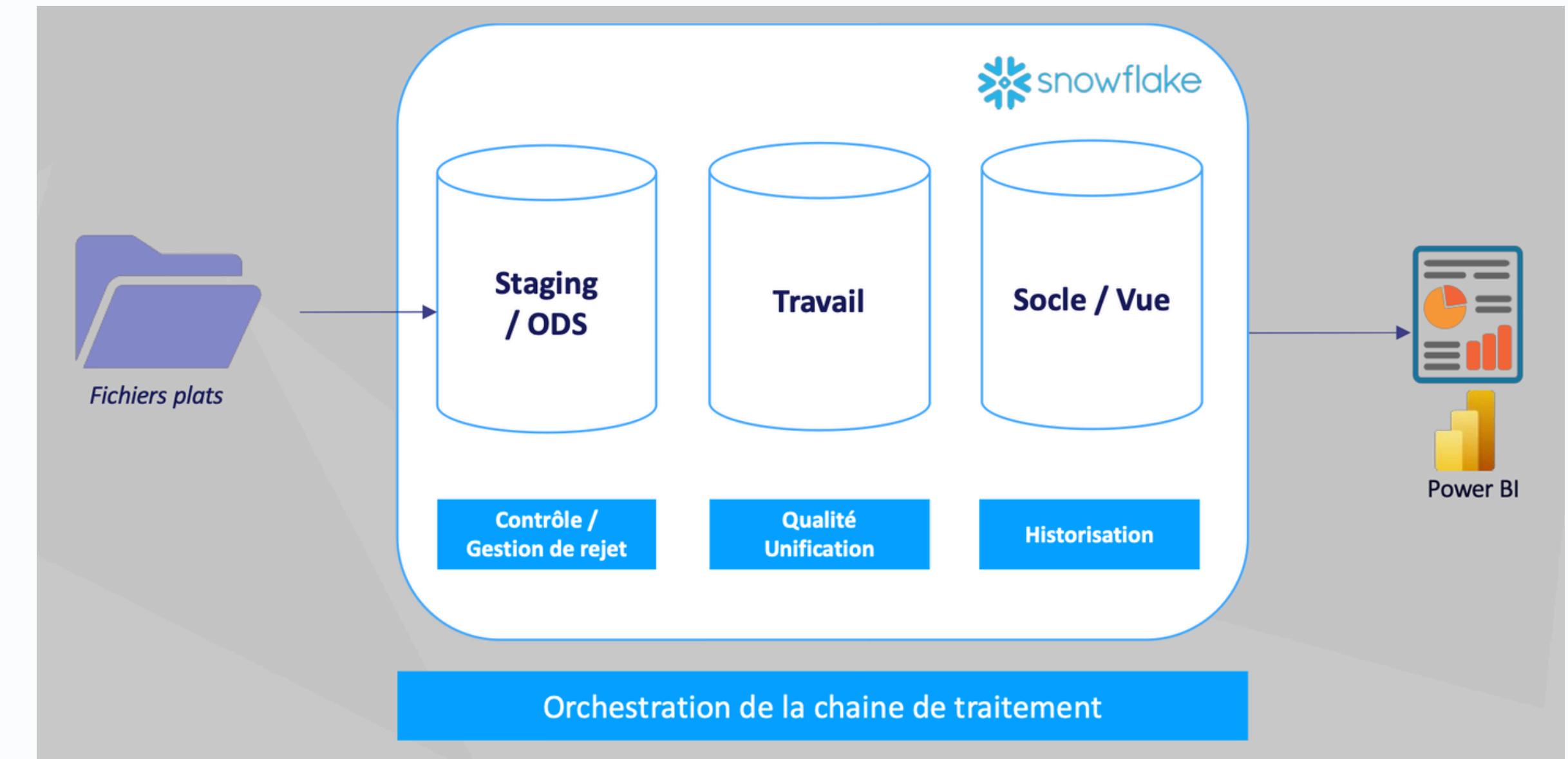
0% 0 total points 0 completed points | 6 open tasks 2 closed tasks ↗ | 0 cocaine doses

ZOOM: Detailed

NEW	IN PROGRESS	READY FOR TEST	CLOSED
#3 2.1.2 - Créer les scripts SQL de création de tables	#6 2.1.5 - Tester plusieurs exécutions pour garantir qu'il n'y a jamais d'erreur	#5 2.1.4 - Gérer les logs (install_sid_*.log) dans le script install_sid.py	#2 2.1.1 -Rédiger un script SQL ou Python pour créer les bases si elles n'existent pas
#7 2.2.1 - Écrire les scripts SQL d'alimentation des tables STG	#8 2.2.2 - Développer le script launch_load_sid.py	#9 2.2.3 - Tester que les	#4 2.1.3 - Développer install_sid.py

# 1. Mise en place de l'environnement

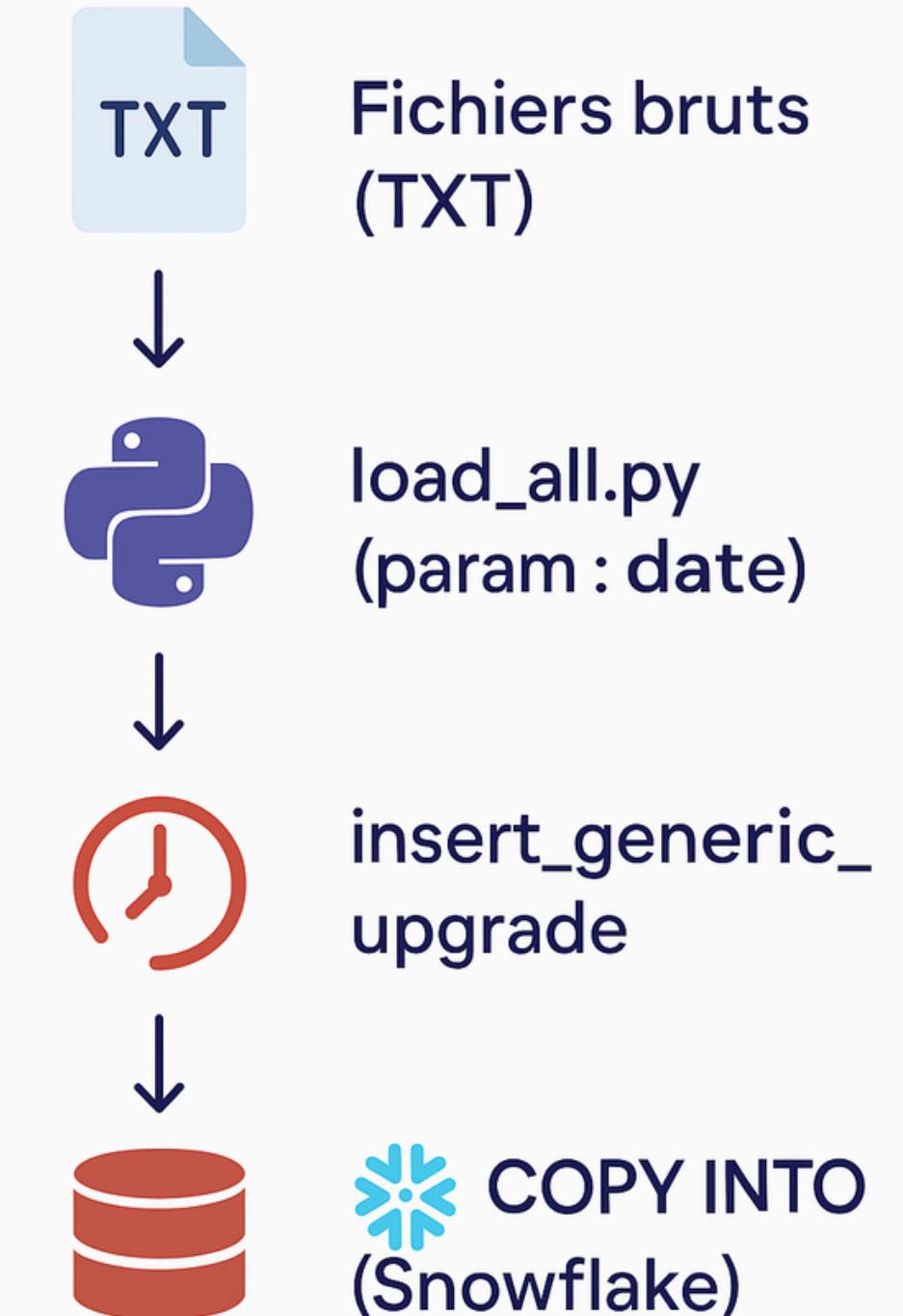
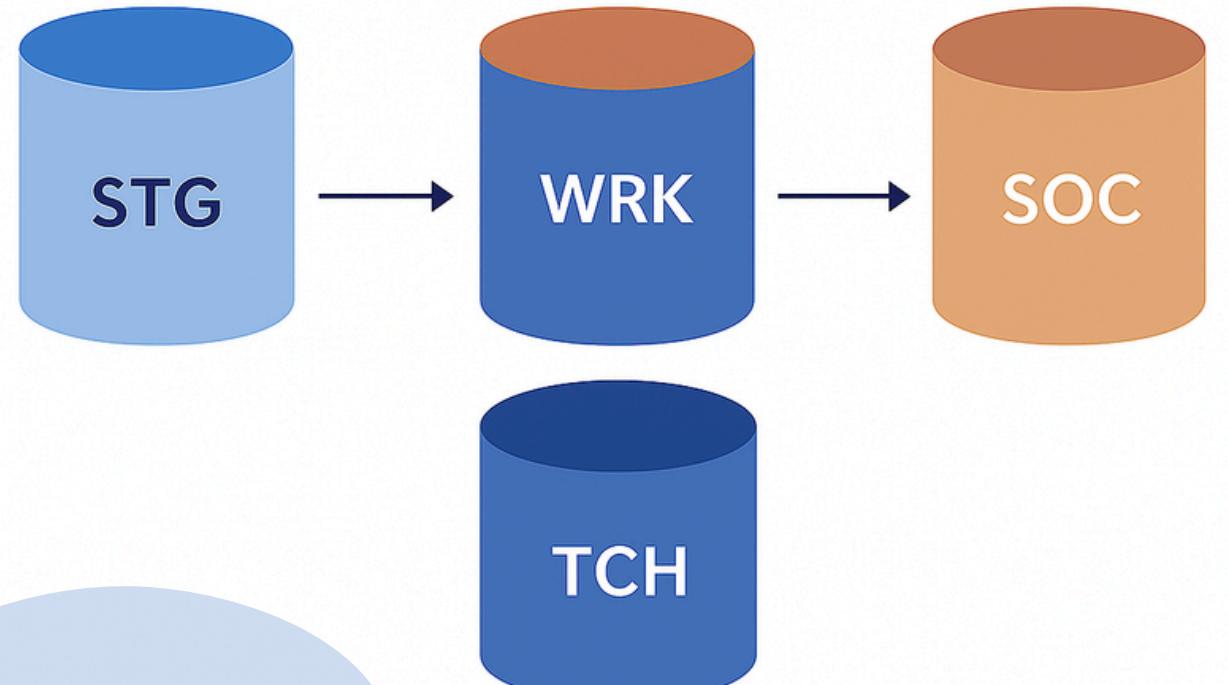
- Étude du Mapping fournit
- PlantUML : UML
- STG, WRK, SOC, TCH



- Nous avons pu poser les bases avant de commencer l'installation du SID et le chargement des données

## 2. Installation du SID et ingestion des données

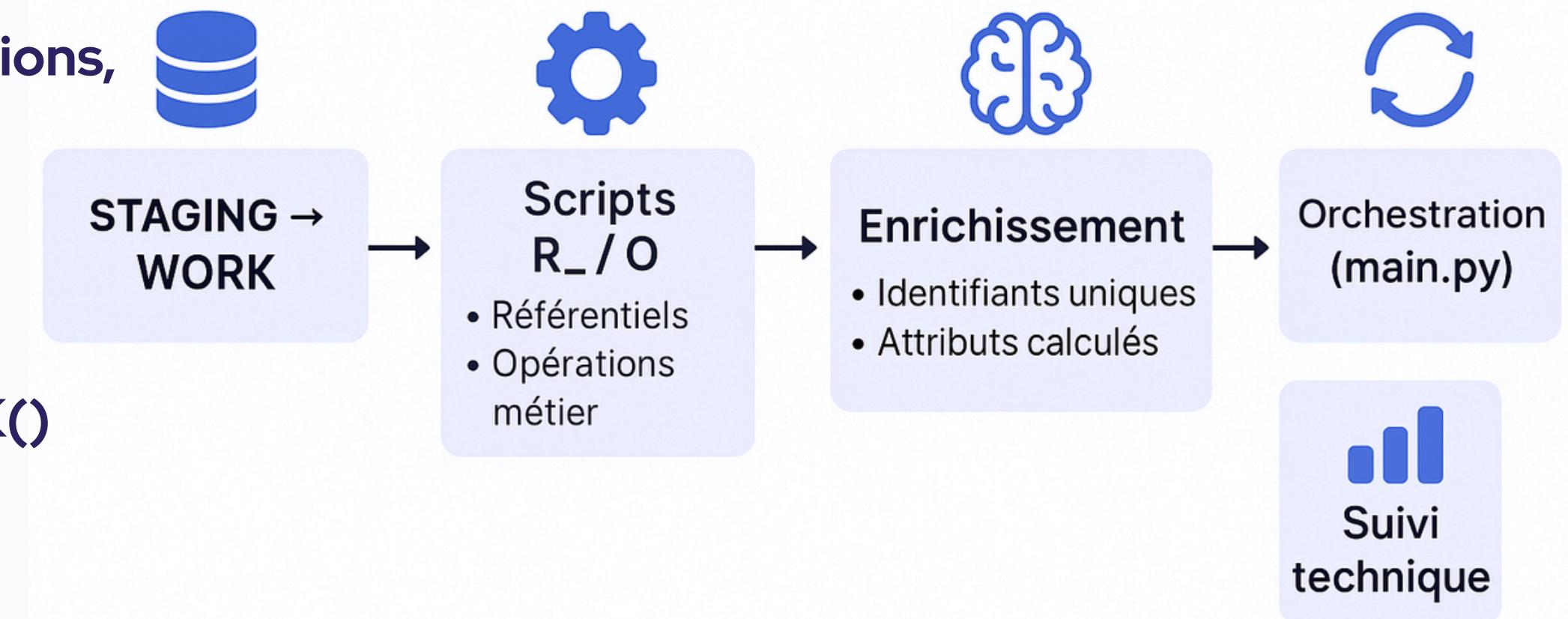
- **Création automatisée** des bases avec `install_sid.py`
- Tables **STG & WRK recréées** à chaque exécution
- **insert\_generic\_upgrade** pour insertion optimisée (vs méthode lente)
- **Fichiers log** générés automatiquement pour chaque exécution



# 3. Transformation et enrichissement des données

## Scripts SQL

- **R\_** pour les référentiels (**patients, chambres, etc.**)
- **O\_** pour les opérations métier (**consultations, hospitalisations...**)



## Enrichissement des données en WRK

- Identifiants uniques via **DENSE\_RANK()**

## Orchestration via main.py

- **run\_id**: chaque exécution complète
- **exec\_id**: chaque script individuel

# 3. Transformation et enrichissement des données

**Contrôle qualité** (cleaners.py), on vérifie :

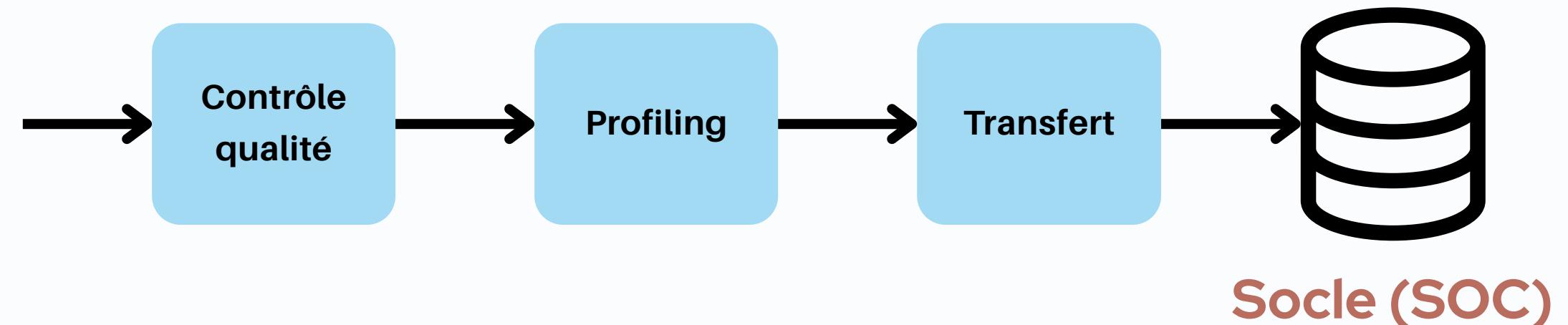
- la présence de valeurs manquantes ;
- la présence de doublons ;
- les types des colonnes ;
- les valeurs aberrantes.

**Profiling** (profiling\_wrk\_complet.sql), on relève :

- le minimum et le maximum ;
- la moyenne ;
- le nombre de valeurs nulles.

**Transfert dans la zone SOC** (launch\_load\_soc.py) :

- insertion des données dans les tables cibles ;
- ordre précis pour respecter les dépendances (e.g. *patient* avant *consultations*) ;
- traçabilité des chargements avec l'attribution d'un *exec\_id* dans la table *SUIVI\_TRAITEMENT*.



# 3. Transformation et enrichissement des données

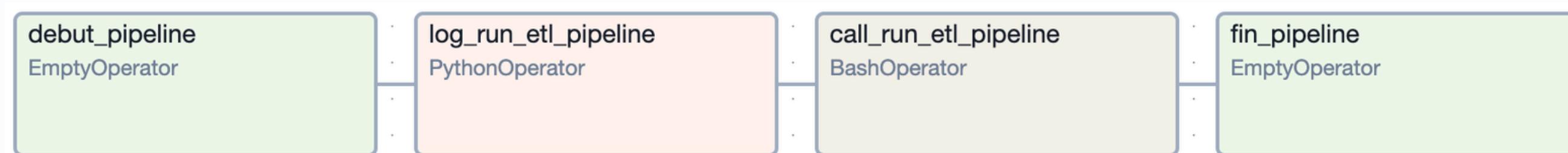
Objectif : **automatiser le chargement** des données.

Création de deux DAGs Airflow :

- **dag\_installation.py** pour la création de la base de données ;
- **dag\_chargement.py** pour le chargement des données par jour.

Chargement des données :

- **format AAAAMMMJJ** pour la date (e.g. 20240429) ;
- transmission du *timestamp* d'exécution du DAG comme **run\_id** (entier) ;
- exécution du fichier **main.py** avec un BashOperator ;
- le DAG peut être configuré pour s'exécuter tous les jours.



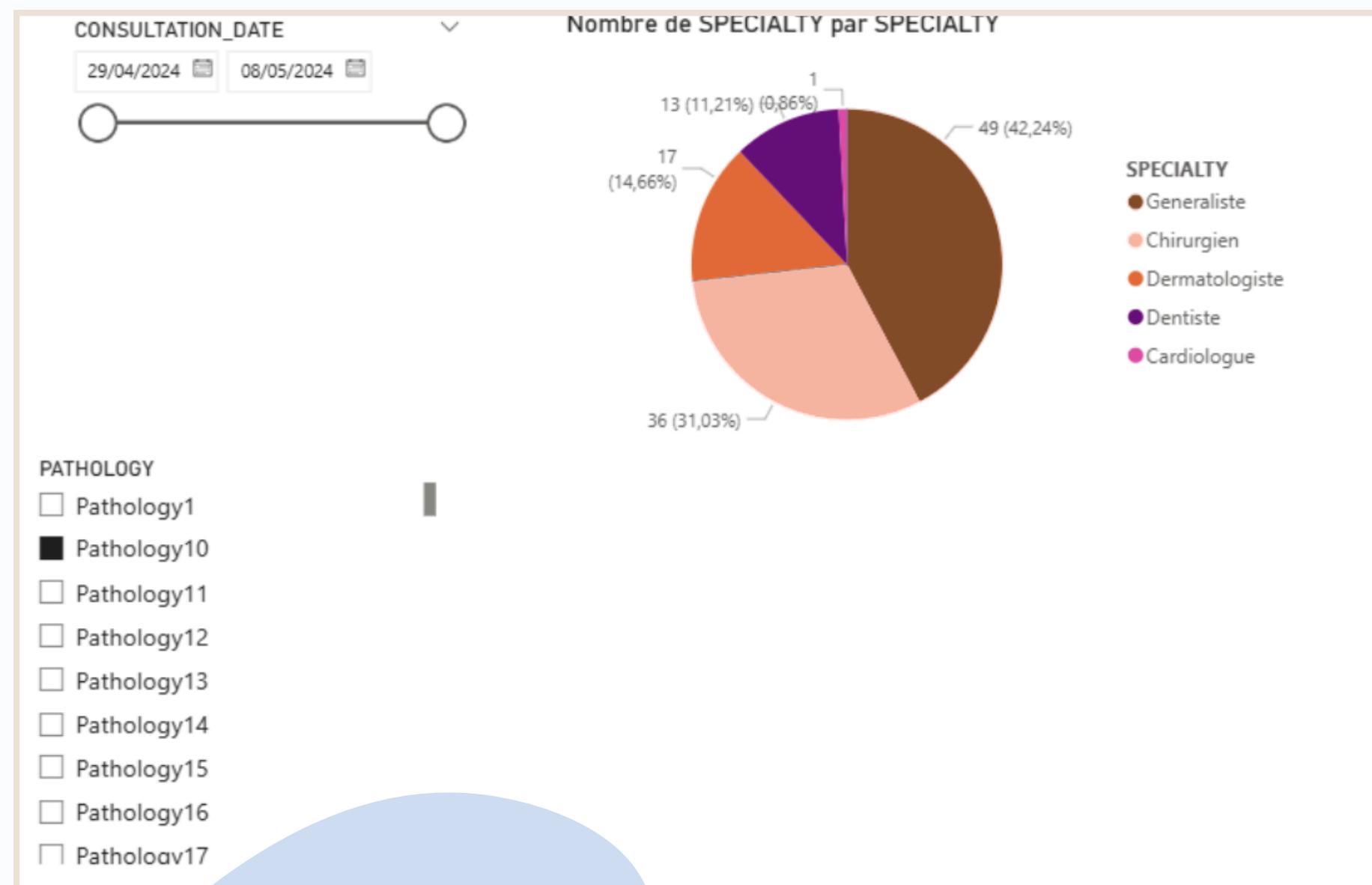
Tâches du DAG de chargement

# 4. Analyse et visualisation des données

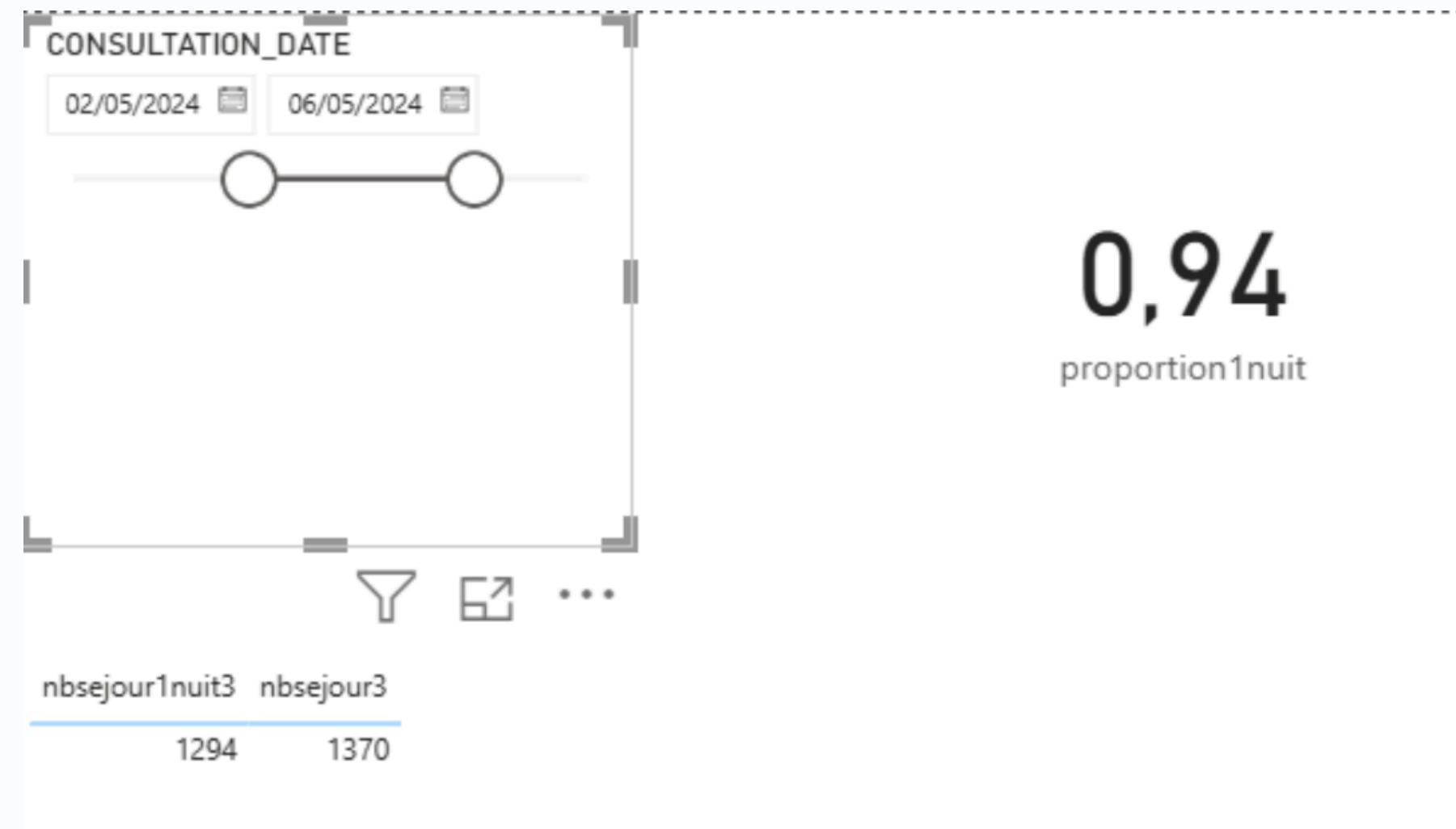


# 4. Analyse et visualisation des données

Quelle est la proportion de médecins (par spécialité) qui ont diagnostiqué une certaine Pathologie durant une certaine période ?



Quelle est la proportion de patients hospitalisés qui ont été restés au moins une nuit durant une certaine période ?



# Conclusion



## Réalisations

- **Projet réalisé avec succès**, toutes les fonctionnalités sont en place.
- **Équipe bien organisée**, travail réparti équitablement.



## Apports du projet

- **Découverte de nouveaux outils** : Airflow, Snowflake, Power BI.
- **Expérience précieuse** avec des technologies utilisées en entreprise.



## Limites et difficultés

- **Mise en place des outils parfois complexe** et technique.
- **Objectifs initiaux flous**, nécessitant des échanges pour clarification.



## Perspectives

- **Évaluer la performance** du code et optimiser les traitements.
- Intégrer des jeux de données plus réalistes et variés.
- Améliorer le tableau de bord Power BI pour des analyses plus poussées.

**Merci pour votre écoute !**